UNIVERSIDAD ADOLFO IBÁÑEZ

Tarea 1:
Sistemas Bioinspirados,
Algoritmos Genéticos aplicados a Travel Salesman Problem (TSP) - Shortest Path in Naval Environment.

(Acceso al GitHub MIA_SistemasBioinspirados de este informe en siguiente enlace)

GitHub

Acceso al notebook Jupyter de la tarea Nº 1 en siguiente enlace.

Estudiante:
Eduardo Carrasco
Argenis Chirinos
Gustavo Jara

Profesor:
Ricardo Contreras

Curso:
Sistemas Bioinspirados. MIA, 2022.

Tabla de contenido

I. Tópico específico - Algoritmos genéticos:

Desarrollo **de la tarea** **:** Grupal.
Evaluación **:** Escala de 1 a 7.
Entrega **de la tarea** **:** Hasta 28 de julio.

A. Objetivo:

Mostrar el nivel de comprensión alcanzado en las clases, a partir de un ejemplo que considere los aspectos relevantes de los algoritmos genéticos.

B. Criterios de evaluación:

1. Respecto de los resultados de aprendizaje:
   a. Se comprenden cabalmente los componentes necesarios para el desarrollo de una solución.
   b. Es posible identificar una aplicación
   c. Existe una descripción del programa construido. La documentación es pertinente.
2. Respecto de los logros:
   a. Se aplican correctamente los criterios para determinar los parámetros típicos de una aplicación demostrativa.
   b. Los resultados obtenidos son interpretados de manera crítica.
   c. Las metas elegidas son realizables.
3. Calidad del informe:
   a. Estructura (considerar Introducción, Antecedentes, Criterios principales de la solución).
   b. Capacidad de síntesis.
   c. Calidad de las conclusiones.
4. Ejecución correcta del programa.

II. Desarrollo de la tarea:

Se cuenta con la carta Nº 7000 del Servicio Hidrográfico y Oceanográfico de la Armada (SHOA) que permite la navegación por el área Sur desde Bahía Corral (Referencia Valdivia) hasta Boca del Guafo (Referencia Isla Guafo al Sur de Isla Chiloé).
En la siguiente figura, se muestra la Carta de Navegación, donde cada cuadro negro representa el punto que indica la ruta comercial para dirigirse a parte de los puertos/canales/bahías de la Isla de Chiloé.

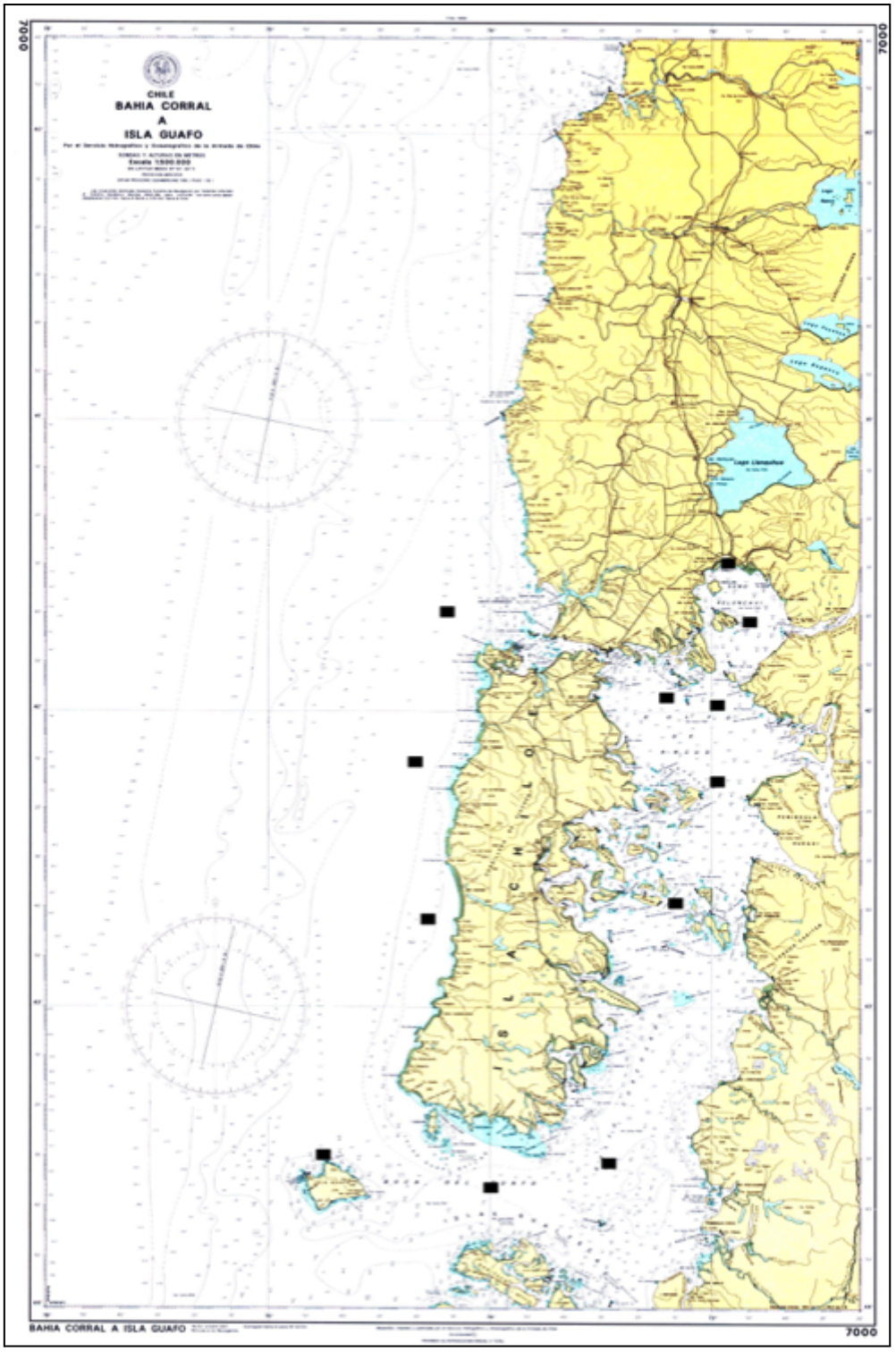

Figura Nº 1: Carta Nº 7000 con puntos GPS propuestos.

Para lo anterior, se utilizó Google Earth, objeto determinar la posición GPS de cada punto (cuadrado negro) obteniendo un kml/csv que se almacenó como archivo "PUNTO_PRUEBA".csv del mismo root del notebook.

III. Puntos GPS determinados en base a una ruta de navegación:

Utilizando el archivo generado anteriormente, determinaremos las distancias en millas Náuticas, tal como se utilizan en el ambiente marítimo.

Código se observa en Anexo A de este documento.

	X	Y	Name	description
0	-72.921258	-41.490495	P1	PMontt
1	-72.828036	-41.694437	P2	Reloncavi
2	-73.016551	-41.961199	P3	Ancud
3	-72.993615	-42.273183	P4	Mechuque
4	-73.171205	-42.657329	P5	Desertores
5	-73.399562	-43.404184	P6	Quellon
6	-73.988938	-43.531862	P7	Inio
7	-74.742380	-43.513953	P8	Guafo
8	-74.274367	-42.826613	P9	Cucao
9	-74.236592	-42.177675	P10	Chepu
10	-74.124650	-41.699442	P11	Chacao
11	-73.260156	-41.864961	P12	Calbuco

Figura Nº 2: Tabla de puntos GPS propuestos, obtenidos desde archivo csv.

A. Determinación de distancias usando GEOPY:

Se debe utilizar la librería geopy, objeto determinar de mejor manera las distancias. En caso de no utilizar la librería descrita, se debe determinar la distancia utilizando la fórmula Harvesine, que se detalla a continuación:

$$d = 2r\,arcsin\left(\sqrt{hav(\varphi_2 - \varphi_1) + cos(\varphi_1)cos(\varphi_2)hav(\lambda_2 - \lambda_1)}\right)$$

$$d = 2r\,arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + cos(\varphi_1)cos(\varphi_2)sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Considerar que la librería permite el cálculo de distancia en kilómetros, por lo cual, se debe dividir por 1.852 para obtener la tabla de distancias en millas náuticas.

Código se observa en Anexo A de este documento.

Name	PMontt	Reloncavi	Ancud	Mechuque	Desertores	Quellon	Inio	Guafo	Cucao	Chepu	Chacao	Calbuco
PMontt	0.0	12.9	28.6	47.1	70.9	116.7	131.3	145.8	100.3	72.0	55.6	27.1
Reloncavi	12.9	0.0	18.1	35.5	59.7	105.6	121.6	138.2	93.6	69.4	58.3	21.9
Ancud	28.6	18.1	0.0	18.7	42.3	88.2	103.6	120.4	76.3	56.0	52.1	12.3
Mechuque	47.1	35.5	18.7	0.0	24.4	70.2	87.3	107.2	65.8	55.7	61.2	27.2
Desertores	70.9	59.7	42.3	24.4	0.0	45.9	63.6	86.1	49.8	55.4	71.5	47.7
Quellon	116.7	105.6	88.2	70.2	45.9	0.0	26.9	59.0	51.8	82.3	107.2	92.5
Inio	131.3	121.6	103.6	87.3	63.6	26.9	0.0	32.9	44.1	82.0	110.1	105.1
Guafo	145.8	138.2	120.4	107.2	86.1	59.0	32.9	0.0	46.1	83.2	112.2	118.7
Cucao	100.3	93.6	76.3	65.8	49.8	51.8	44.1	46.1	0.0	39.0	67.9	73.2
Chepu	72.0	69.4	56.0	55.7	55.4	82.3	82.0	83.2	39.0	0.0	29.1	47.5
Chacao	55.6	58.3	52.1	61.2	71.5	107.2	110.1	112.2	67.9	29.1	0.0	40.1
Calbuco	27.1	21.9	12.3	27.2	47.7	92.5	105.1	118.7	73.2	47.5	40.1	0.0

Figura Nº 3: Tabla de distancias (geodésicas), en millas náuticas, de todos los puntos de interés.

B. Formación y verificación coordenadas geográficas:

Efectuado lo anterior, se genera un nuevo df objeto sea posicionado en folium y geopandas.

Código se observa en Anexo A de este documento.

	Point Name	Latitude	Longitude	Description	geometry
0	P1	-41.490495	-72.921258	PMontt	POINT (-72.92126 -41.49049)
1	P2	-41.694437	-72.828036	Reloncavi	POINT (-72.82804 -41.69444)
2	P3	-41.961199	-73.016551	Ancud	POINT (-73.01655 -41.96120)
3	P4	-42.273183	-72.993615	Mechuque	POINT (-72.99361 -42.27318)
4	P5	-42.657329	-73.171205	Desertores	POINT (-73.17121 -42.65733)
5	P6	-43.404184	-73.399562	Quellon	POINT (-73.39956 -43.40418)
6	P7	-43.531862	-73.988938	Inio	POINT (-73.98894 -43.53186)
7	P8	-43.513953	-74.742380	Guafo	POINT (-74.74238 -43.51395)
8	P9	-42.826613	-74.274367	Cucao	POINT (-74.27437 -42.82661)
9	P10	-42.177675	-74.236592	Chepu	POINT (-74.23659 -42.17767)
10	P11	-41.699442	-74.124650	Chacao	POINT (-74.12465 -41.69944)
11	P12	-41.864961	-73.260156	Calbuco	POINT (-73.26016 -41.86496)

Figura Nº 4: Tabla de puntos de interés en función de Lon/Lat.

Se realiza una comprobación geográfica de los puntos, utilizando el plano GPS, objeto observar si están correctamente referenciados.

Código se observa en Anexo A de este documento.

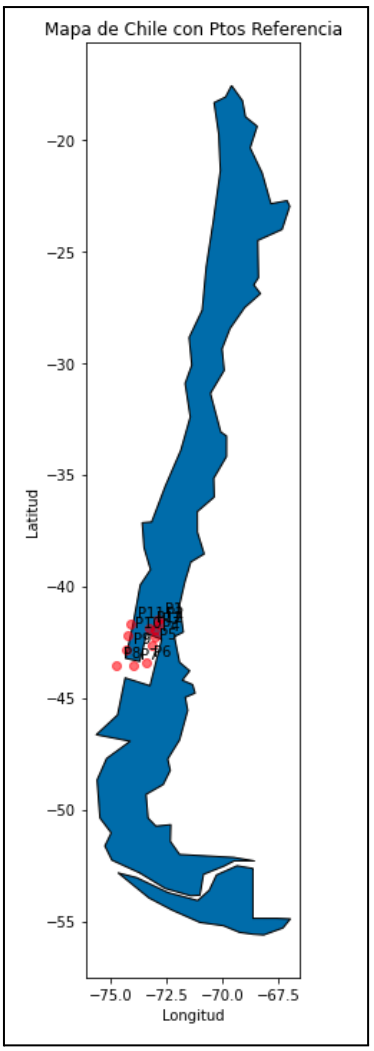

Figura Nº 5: Referencia geográfica de puntos agregados en función de Lon/Lat.

Se utiliza la librería folium objeto acceder al mapa y puntos geográficos.

Código se observa en Anexo A de este documento.

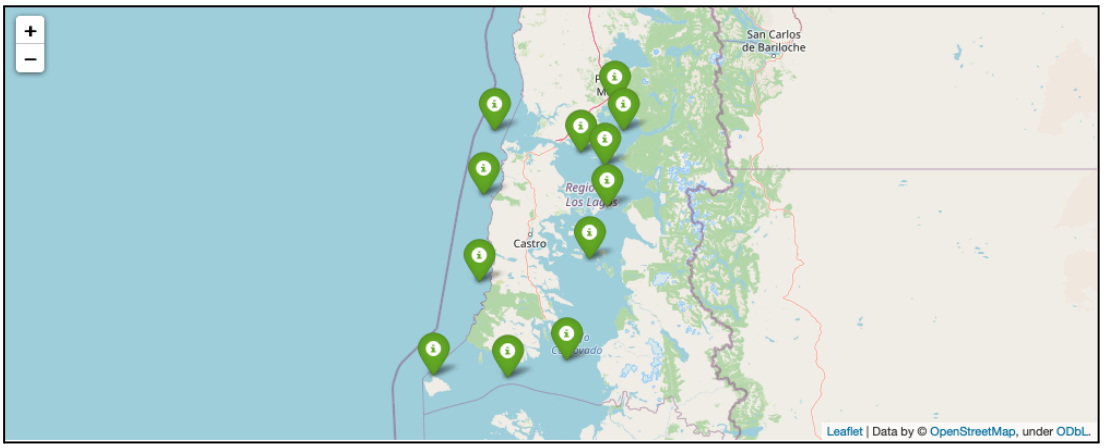

Figura Nº 6: Chequeo de puntos en folium.

Como se observa en la imagen anterior los puntos fueron correctamente generados. Al hacer click en cada uno de los puntos azules, se observa la información (descripción) relacionada a su posición geográfica.

IV. TSP a través de algoritmos genéticos:

Previo a la determinación de la ruta más corta, tenemos como referencia un plano cartesiano compuesto por toda la tierra (GPS), en latitud (eje y) y longitud (eje x), tal como se muestra en la siguiente figura:

Figura Nº 7: Gráfico coordenadas geográficas mundiales.

Código se observa en Anexo A de este documento.

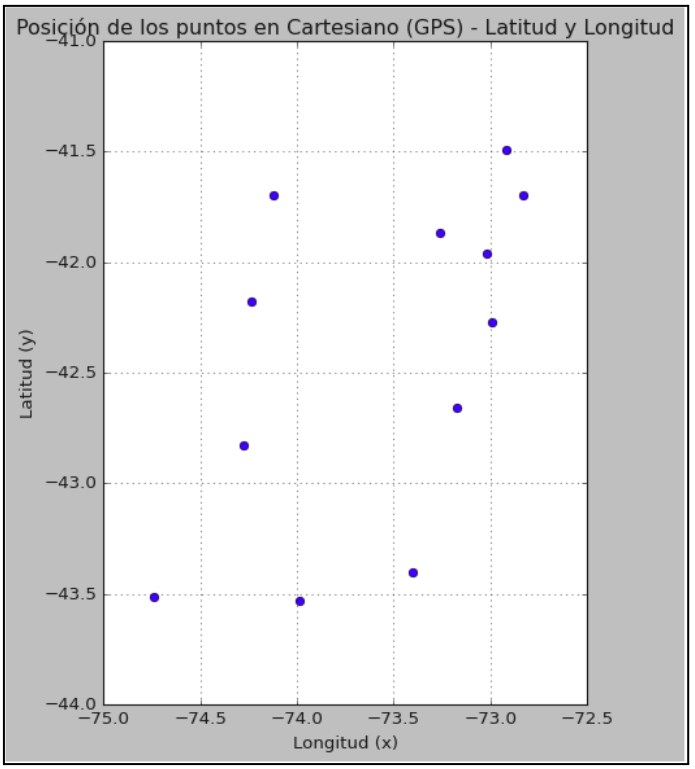

Figura Nº 8: Gráfico de puntos de interés en función de latitud y longitud.

A. Confección y selección de variables:

Antes de comenzar la definición de variables, se debe mencionar que el algoritmo genético es una técnica de optimización, que a través de una función de evaluación (fitness), permite determinar el mínimo/máximo.

En este caso particular del TSP, se requiere encontrar el mínimo, es decir, la mínima ruta (ruta más corta), entre todos los puntos.

Para lograr lo anterior, se debe confeccionar el *array* con puntos geográficos, lo cual conformará el cromosoma.

Código se observa en Anexo A de este documento.

```
array(['PMontt', 'Reloncavi', 'Ancud', 'Mechuque', 'Desertores',
       'Quellon', 'Inio', 'Guafo', 'Cucao', 'Chepu', 'Chacao', 'Calbuco'],
      dtype=object)
```

Figura Nº 9: Cromosoma creado en función de puntos geográficos.

Todas las combinaciones del array anterior (12^{12} cromosomas) es lo que se llama **población**, que forma el subconjunto de soluciones posibles, que por la cantidad de genes que posee el cromosoma, lo hace ser una solución difícil de calcular si no fuera por estos algoritmos.

La codificación de cada gen, en este caso es de “valor secuencial”, es decir, tiene un número del 0 al 11 representada por el nombre del punto geográfico.

La etiqueta (información) que contiene cada **gen** (0 - 11), se llama **alelo**, para este ejemplo el alelo del gen 0 es “Pmontt”.

- 0 = PMontt.
- 1 = Reloncaví.
- 2 = Ancud.
- 3 = Mechuque.
- 4 = Desertores.
- 5 = Quellon.
- 6 = Inio.
- 7 = Guafo.
- 8 = Cucao.
- 9 = Chepu.
- 10 = Chacao.
- 11 = Calbuco.

Confección de *array* con matriz de distancias, lo cual permitirá determinar el fitness en el siguiente paso.

Código se observa en Anexo A de este documento.

```
array([[  0. ,  12.9,  28.6,  47.1,  70.9, 116.7, 131.3, 145.8, 100.3,
         72. ,  55.6,  27.1],
       [ 12.9,   0. ,  18.1,  35.5,  59.7, 105.6, 121.6, 138.2,  93.6,
         69.4,  58.3,  21.9],
       [ 28.6,  18.1,   0. ,  18.7,  42.3,  88.2, 103.6, 120.4,  76.3,
         56. ,  52.1,  12.3],
       [ 47.1,  35.5,  18.7,   0. ,  24.4,  70.2,  87.3, 107.2,  65.8,
         55.7,  61.2,  27.2],
       [ 70.9,  59.7,  42.3,  24.4,   0. ,  45.9,  63.6,  86.1,  49.8,
         55.4,  71.5,  47.7],
       [116.7, 105.6,  88.2,  70.2,  45.9,   0. ,  26.9,  59. ,  51.8,
         82.3, 107.2,  92.5],
       [131.3, 121.6, 103.6,  87.3,  63.6,  26.9,   0. ,  32.9,  44.1,
         82. , 110.1, 105.1],
       [145.8, 138.2, 120.4, 107.2,  86.1,  59. ,  32.9,   0. ,  46.1,
         83.2, 112.2, 118.7],
       [100.3,  93.6,  76.3,  65.8,  49.8,  51.8,  44.1,  46.1,   0. ,
         39. ,  67.9,  73.2],
       [ 72. ,  69.4,  56. ,  55.7,  55.4,  82.3,  82. ,  83.2,  39. ,
          0. ,  29.1,  47.5],
       [ 55.6,  58.3,  52.1,  61.2,  71.5, 107.2, 110.1, 112.2,  67.9,
         29.1,   0. ,  40.1],
       [ 27.1,  21.9,  12.3,  27.2,  47.7,  92.5, 105.1, 118.7,  73.2,
         47.5,  40.1,   0. ]])
```

Figura Nº 10: Tabla de distancias en array.

B. Algoritmo Genético (definición):

Objeto desarrollar el Algoritmo Genético, previamente se deben reconocer los siguientes 5 pasos:

1. Generación de la población inicial (***Population generation***): donde se genera una población randomizada de cromosomas.
2. Evaluación del Fitness (***Fitness evaluation***): se calcula el fitness de todos los cromosomas en la población.
3. Selección de los Padres/pares (***Parent selection***): donde ciertos cromosomas son seleccionados para ser trasladados a la siguiente generación en base al elitismo (más fuerte / el mejor).
4. Cruzamiento (***Crossover***): donde secciones del mejor match de cromosomas son combinados para producir descendencia.
5. Mutación (***Mutation***): donde los cromosomas son modificados para producir una nueva solución.

Como se señala en la referencia [2], los Algoritmos genéticos están basados en el principio de *"survival of the fittest"* (Sobrevive el más fuerte). Cada solución es representada por un Cromosoma (***Chromosome***), que consiste en una secuencia de genes que representan la solución al problema.

Los Algoritmos Genéticos (GA) operan en una población de soluciones posibles, estas son manipuladas en base a iteraciones a las cuales se les llama generaciones (**generations**).

Para determinar mejores soluciones progresivamente, el algoritmo hace match (pairs) de chromosomes pares, produce descendencia (**offspring**) o de alguna forma genera mutaciones (**mutation**) de manera alterna en los chromosomes previamente creados.

Finalmente, cuando la población obtiene la mejor solución (la descendencia no es diferente de sus padres), el algoritmo finaliza.

C. Parámetros de Inicialización:

Se deben setear los parámetros por sobre los cuales, el algoritmo efectuará los cálculos, en este caso se decidión ingresar los siguientes valores:

1. Cantidad de población Inicial (num_population) = 25.
2. Número de generaciones (num_generations) = 500.
3. Probabilidad de cruzamiento (prob_crossover) = 60%.
4. Probabilidad de mutación (prob_mutation) = 50%.

Código se observa en Anexo A de este documento.

En conjunto los parámetros de inicialización, se definen las siguientes funciones:

1. Cromosomas.
2. Evaluación del cromosoma.
3. Cálculo de Fitness.
4. Evaluación de la mejor solución (fitness/orden) en función de la generación (output).

Código se observa en Anexo A de este documento.

Efectuado lo anterior y verificado el fitness de cada generación, se puede determinar donde se efectuó la convergencia. Lo anterior, se puede graficar en la siguiente figura, que demuestra que en la generación 145 se produce la convergencia que lleva a la optimización de la ruta (de acuerdo a los parámetros de inicialización seteados anteriormente).

Código se observa en Anexo A de este documento.

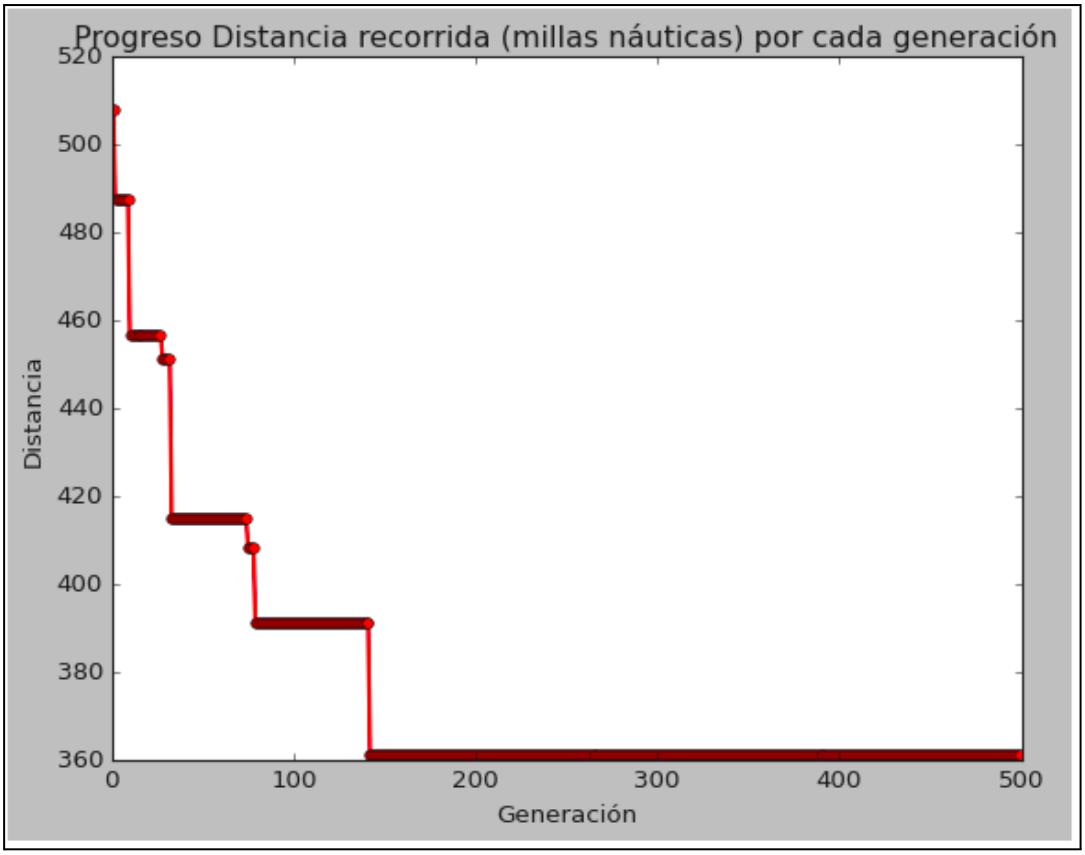

Figura Nº 11: Gráfico fitness en función de la generación.

Con lo anterior, de manera de representar la mejor ruta, se puede obtener el siguiente output que representa el crosomona de solución y el mejor fitness (minimización):

Código se observa en Anexo A de este documento.

```
La mejor opción es: [5, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3, 4] | Mínima distancia recorrida: 361.2
```

Figura Nº 12: Resultado final de la implementación de AG.

Para este caso particular, el criterio de terminación se decidió establecer en observar que no hay mejoras en optimalidad después de algunas iteraciones (observado en el gráfico).

Finalmente, podemos graficar cada punto en el plano cartesiano y establecer la ruta que sigue la generación de convergencia para optimizar la ruta.
Código se observa en Anexo A de este documento.

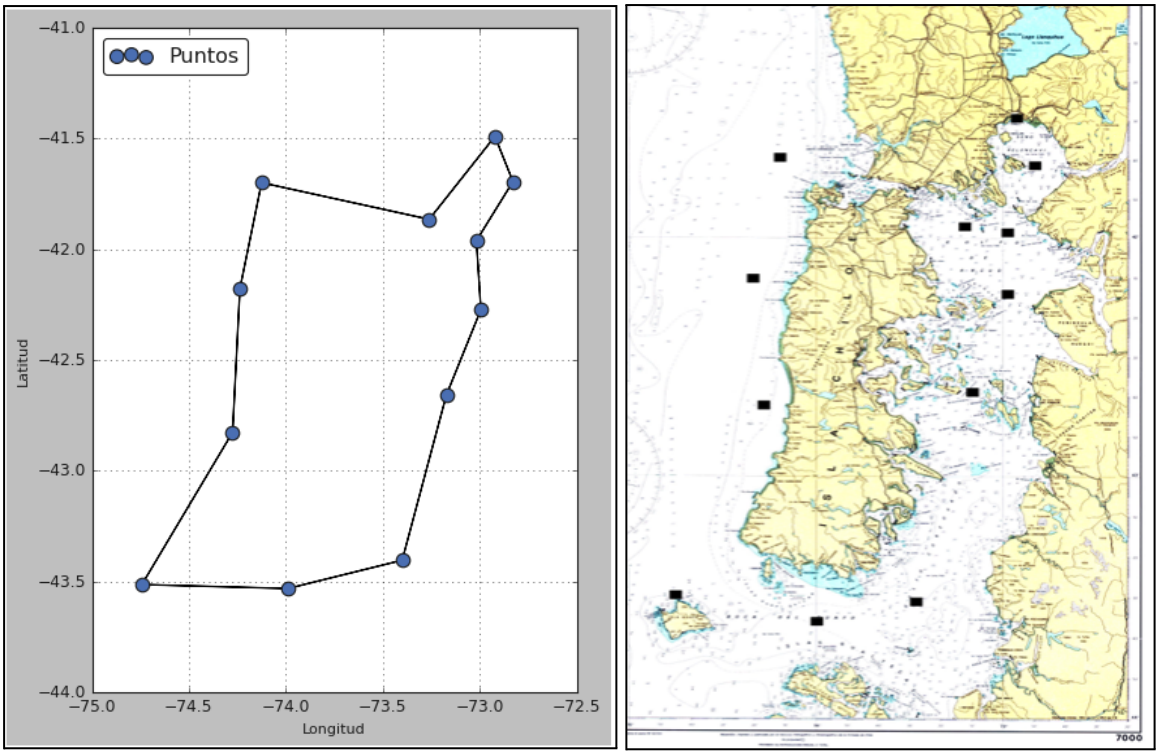

Figura Nº 13: Ruta más corta y comparación con carta Nº 7000.

La distancia por carta total (ruta más corta) pasando por cada uno de los puntos es de 361.2 [mn].

V. Conclusiones y comentarios finales:

A partir de los resultados obtenidos, podemos determinar que el algoritmo genético es capaz de determinar la ruta más corta considerando distancias geodésicas (GPS) en una ruta marítima, considerando pasar por cada punto de control/interés asignado. Lo anterior fue comprobado al sumar las distancias entre cada punto desde "Pmontt" a "Pmontt", obteniendo 362.1mn; mismo resultado obtenido por el algoritmo genético.

A pesar de ser una implementación básica, este demuestra la practicidad de determinación de rutas utilizando este método como mecanismo de optimización, considerando una buena técnica para la cantidad de combinaciones posibles.

VI. Referencias:

[A]. Browniee, Jason (2021). Simple Genetic Algorithm From Scratch in Python. Machine Learning Mastery. [En línea]. Disponible en: https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/, [Acceso el 22 de julio de 2022].

[B]. Imperial College of London (S.F.). Genetic Algorithm. [En línea]. Disponible en: https://transport-systems.imperial.ac.uk/tf/60008_21/n6_04_travelling_salesman_problem-genetic_algorithm.html, [Acceso el 22 de julio de 2022].

[C]. Kasumi, Ijn (2020). Ant colony optimization in the travel salesman problem (TSP). Medium. [En línea]. Disponible en: https://medium.com/@sakamoto2000.kim/ant-colony-optimization-aco-in-the-travel-salesman-problem-tsp-54f83ccd9eff, [Acceso el 22 de julio de 2022].

[D]. Kim, J., & Kim, S. K. (2019). Genetic Algorithms for Solving Shortest Path Problem in Maze-Type Network with Precedence Constraints. Wireless Personal Communications: An International Journal, 105(2), 427–442. https://doi.org/10.1007/s11277-018-5740-3

[E]. Reguant, Roc (2022). How to Implement a Traveling Salesman Problem Genetic Algorithm in Python. Level Up Coding (Medium). [En línea]. Disponible en: https://levelup.gitconnected.com/how-to-implement-a-traveling-salesman-problem-genetic-algorithm-in-python-ea32c7bef20f, [Acceso el 22 de julio de 2022].

[F]. Stoltz, Eric (2018). Evolution of a salesman: A complete genetic algorithm tutorial for Python. Towards Data Science (Medium). [En línea]. Disponible en: https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35, [Acceso el 22 de julio de 2022].

Anexo “A”

Código de Python 3

IDE: Jupyterlab versión 3.1.7. (2021)
web site https://jupyter.org/install

Código implementación:

```
# pip install geopy folium
```

```
import pandas
import pandas as pd
import numpy as np
df1 = pandas.read_csv('PUNTOS_PRUEBA.csv') #sep default de pandas es
coma.
print(df1)
```

Cálculo de matriz utilizando librería geopy:

```
import numpy as np
from geopy import distance
from geopy.distance import lonlat, distance
empty_matrix = np.zeros((12,12))

for i in range(12):
    for j in range(12):
        if i != j:
            #print(i,j,df2.iloc[i,1],df2.iloc[j,1],distance(lonlat(*df2.iloc[i,0:2]),
            #lonlat(*df2.iloc[j,0:2])).km)
            empty_matrix[i,j] = ((distance(lonlat(*df1.iloc[i,0:2]),
lonlat(*df1.iloc[j,0:2])).km)/1.852)

empty_matrix
df_2 = empty_matrix
df_mn = pd.DataFrame(df_2, columns = df1.iloc[:,3])# convertir NumPy Array a
Pandas DF
df_mn.insert(0,'Name', df1.iloc[:,3])
df_mn1=round(df_mn,1)
df_mn1
```

Colocación de puntos geográficos en plano GPS:

```
GPSposition = {'Point Name': df1.iloc[:,2],
          'Latitude': df1.iloc[:,1],
          'Longitude': df1.iloc[:,0],
         'Description': df1.iloc[:,3]}
frame1 = pd.DataFrame(GPSposition)

import geopandas
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
from shapely.geometry import Point

%matplotlib inline

gdf = geopandas.GeoDataFrame(frame1.iloc[:,0:4],
                             geometry=geopandas.points_from_xy(frame1.Longitude,
frame1.Latitude))
print(gdf)

fig, gax = plt.subplots(figsize=(5,12))
world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))
world.query("name == 'Chile'").plot(ax=gax, edgecolor='black')

gdf.plot(ax=gax, color='red', alpha = 0.5)

gax.set_xlabel('Longitud')
gax.set_ylabel('Latitud')
gax.set_title('Mapa de Chile con Ptos Referencia')

gax.spines['top'].set_visible(True)
gax.spines['right'].set_visible(True)

for x, y, label in zip(gdf['geometry'].x, gdf['geometry'].y, gdf['Point Name']):
    gax.annotate(label, xy=(x,y), xytext=(5,5), textcoords='offset points')

plt.show()
```

Colocación en plano gps utilizando librería folium:

```
import folium
m = folium.Map(tiles='OpenStreetMap', width=1000,
               height=400,location=[gdf.Latitude.mean(),
               gdf.Longitude.mean()], zoom_start=7, min_zoom=3,max_zoom=100)
for index, location_info in gdf.iterrows():
    folium.Marker([location_info["Latitude"], location_info["Longitude"]],
                  popup=location_info["Description"], icon=folium.Icon(color="green",
icon="info-sign")).add_to(m)
m
```

```
fig = plt.figure(figsize=(6,8))
ax = fig.add_subplot()
ax.plot(gdf.iloc[:,2], gdf.iloc[:,1], marker="o", c="blue", linestyle='')
ax.grid(True)
ax.set_xlabel("Longitud (x)")
ax.set_ylabel("Latitud (y)")
ax.set_title("Posición de los puntos en Cartesiano (GPS) - Latitud y Longitud")
plt.style.use("classic")
plt.show()
```

Confección y selección de variables:

```
from deap import base, creator, tools
puntos = df_mn1.iloc[0:13,0].to_numpy()
puntos

dist_matrix = df_mn1.iloc[0:13,1:13].to_numpy()
dist_matrix

num_homes = df_mn1.iloc[0:12,0].to_numpy().size
homes_names = [i for i in range(num_homes)]

homes_coord = gdf.iloc[:,1:3].to_numpy()
homes_coord[:, [1, 0]] = homes_coord[:, [0, 1]] # esto se aplica para efectuar swap
considerando latitud es y.
homes_coord
```

Funciones e inicialización:

```
import copy

np.random.seed(3)

def chromo_create(_homes_names):
    chromo = copy.deepcopy(_homes_names)
    np.random.shuffle(chromo)
    return chromo

def chromo_eval(_dist_matrix, _chromo):
    dist = 0
    for p in range(len(_chromo) - 1):
        _i = _chromo[p]
        _j = _chromo[p+1]
        dist += _dist_matrix[_i][_j]

    dist += dist_matrix[_chromo[-1], _chromo[0]]
    return dist,
```

```
tb = base.Toolbox()
creator.create('Fitness_Func', base.Fitness, weights=(-1.0,))
creator.create('Individual', list, fitness=creator.Fitness_Func)

#inicialización
num_population = 25
num_generations = 500
prob_crossover = .6
prob_mutation = .5

tb.register('indexes', chromo_create, homes_names)
tb.register('individual', tools.initIterate, creator.Individual, tb.indexes)
tb.register('population', tools.initRepeat, list, tb.individual)
tb.register('evaluate', chromo_eval, dist_matrix)
tb.register('select', tools.selTournament)
tb.register('mate', tools.cxPartialyMatched)
tb.register('mutate', tools.mutShuffleIndexes)

population = tb.population(n=num_population)

fitness_set = list(tb.map(tb.evaluate, population))
for ind, fit in zip(population, fitness_set):
    ind.fitness.values = fit
```

Evaluación y output:

```
best_fit_list = []
best_sol_list = []

best_fit = np.Inf

for gen in range(0, num_generations):

    if (gen % 5 == 0):
        print(f'Generacion: {gen:4} | Fitness: {best_fit:.2f}')

    offspring = tb.select(population, len(population), tournsize=3)
    offspring = list(map(tb.clone, offspring))

    for child1, child2 in zip(offspring[0::2], offspring[1::2]):
        if np.random.random() < prob_crossover:
            tb.mate(child1, child2)
            del child1.fitness.values
            del child2.fitness.values

    for chromo in offspring:
        if np.random.random() < prob_mutation:
            tb.mutate(chromo, indpb=0.01)
            del chromo.fitness.values

    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
```

```
    fitness_set = map(tb.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitness_set):
        ind.fitness.values = fit

    population[:] = offspring

    curr_best_sol = tools.selBest(population, 1)[0]
    curr_best_fit = curr_best_sol.fitness.values[0]

    if curr_best_fit < best_fit:
        best_sol = curr_best_sol
        best_fit = curr_best_fit

    best_fit_list.append(best_fit)
    best_sol_list.append(best_sol)

#confección de gráfico de salida
plt.plot(best_fit_list, linewidth=2, color='red', marker='o', markersize=5)
plt.title('Progreso Distancia recorrida (millas náuticas) por cada generación')
plt.ylabel('Distancia')
plt.xlabel('Generación')
plt.style.use("classic")
plt.show()
```

Output con shortest path:

```
plot_size   = 10
plot_width  = 6
plot_height = 8

params = {'legend.fontsize': 'large',
          'figure.figsize': (plot_width,plot_height),
          'axes.grid': True,
          'legend.fancybox': True,
          'axes.labelsize': plot_size,
          'axes.titlesize': plot_size,
          'xtick.labelsize': plot_size*1.1,
          'xtick.labelbottom': True,
          'ytick.labelsize': plot_size*1.1,
          'axes.titlepad': 25}
plt.rcParams.update(params)
plt.rcParams.update(params)

final_sol = best_sol + best_sol[0:1]

plt.scatter(homes_coord[:, 0],
            homes_coord[:, 1],label="Puntos",
            s=plot_size*9,
            cmap='binary',
            zorder = 100);
plt.xlabel("Longitud")
plt.ylabel("Latitud")
plt.legend(loc='upper left')
```

```
lines = []
for p in range(len(final_sol) - 1):
    i = final_sol[p]
    j = final_sol[p+1]
    colour = 'black'
    plt.arrow(homes_coord[i][0], homes_coord[i][1], homes_coord[j][0]
            - homes_coord[i][0], homes_coord[j][1] - homes_coord[i][1],
            color=colour)
```